

Tópico 08 – Funções

Prof. André Gustavo Hochuli

Plano de Aula

- Definição de Função
- Implementações de Funções
- Exercícios



Recursão

- Uma função recursiva é aquela que faz referência a si própria na sua definição
- Um exemplo clássico é o cálculo fatorial:
- $(!4) = (4 \times 3 \times 2 \times 1) \rightarrow 24$
 - $F(4) \rightarrow 4 \times F(3)$

$$\begin{cases} 1 & \text{se } n = 0 \\ n \times (n - 1)! & \text{se } n > 0 \end{cases}$$

Recursão

- Uma função recursiva é aquela que faz referência a si própria na sua definição
- Um exemplo clássico é o cálculo fatorial:
- $(!4) = (4 \times 3 \times 2 \times 1) \rightarrow 24$
 - $F(4) \rightarrow 4 \times F(3)$
 - $F(3) \rightarrow 3 \times F(2)$

$$\begin{cases} 1 & \text{se } n = 0 \\ n \times (n - 1)! & \text{se } n > 0 \end{cases}$$

Recursão

- Uma função recursiva é aquela que faz referência a si própria na sua definição
- Um exemplo clássico é o cálculo fatorial:
- $(!4) = (4 \times 3 \times 2 \times 1) \rightarrow 24$
 - $F(4) \rightarrow 4 \times F(3)$
 - $F(3) \rightarrow 3 \times F(2)$
 - $F(2) \rightarrow 2 \times F(1)$

$$\begin{cases} 1 & \text{se } n = 0 \\ n \times (n - 1)! & \text{se } n > 0 \end{cases}$$

Recursão

- Uma função recursiva é aquela que faz referência a si própria na sua definição
- Um exemplo clássico é o cálculo fatorial:
- $(!4) = (4 \times 3 \times 2 \times 1) \rightarrow 24$
 - $F(4) \rightarrow 4 \times F(3)$
 - $F(3) \rightarrow 3 \times F(2)$
 - $F(2) \rightarrow 2 \times F(1)$
 - $F(1) \rightarrow 1 \times F(0)$

$$\begin{cases} 1 & \text{se } n = 0 \\ n \times (n - 1)! & \text{se } n > 0 \end{cases}$$

Recursão

- Uma função recursiva é aquela que faz referência a si própria na sua definição
- Um exemplo clássico é o cálculo fatorial:
- $(!4) = (4 \times 3 \times 2 \times 1) \rightarrow 24$
 - $F(4) \rightarrow 4 \times F(3)$
 - $F(3) \rightarrow 3 \times F(2)$
 - $F(2) \rightarrow 2 \times F(1)$
 - $F(1) \rightarrow 1 \times F(0)$
 - $F(0) \rightarrow 1$

$$\begin{cases} 1 & \text{se } n = 0 \\ n \times (n - 1)! & \text{se } n > 0 \end{cases}$$

Recursão

- Uma função recursiva é aquela que faz referência a si própria na sua definição
- Um exemplo clássico é o cálculo fatorial:
- $(!4) = (4 \times 3 \times 2 \times 1) \rightarrow 24$
 - $F(4) \rightarrow 4 \times F(3)$
 - $F(3) \rightarrow 3 \times F(2)$
 - $F(2) \rightarrow 2 \times F(1)$
 - $F(1) \rightarrow 1 \times 1 \rightarrow 1$
 - $F(0) \rightarrow 1$

$$\begin{cases} 1 & \text{se } n = 0 \\ n \times (n - 1)! & \text{se } n > 0 \end{cases}$$

Recursão

- Uma função recursiva é aquela que faz referência a si própria na sua definição
- Um exemplo clássico é o cálculo fatorial:
- $(!4) = (4 \times 3 \times 2 \times 1) \rightarrow 24$
 - $F(4) \rightarrow 4 \times F(3)$
 - $F(3) \rightarrow 3 \times F(2)$
 - $F(2) \rightarrow 2 \times 1 \rightarrow 2$
 - $F(1) \rightarrow 1 \times 1 \rightarrow 1$
 - $F(0) \rightarrow 1$

$$\begin{cases} 1 & \text{se } n = 0 \\ n \times (n - 1)! & \text{se } n > 0 \end{cases}$$

Recursão

- Uma função recursiva é aquela que faz referência a si própria na sua definição
- Um exemplo clássico é o cálculo fatorial:
- $(!4) = (4 \times 3 \times 2 \times 1) \rightarrow 24$
 - $F(4) \rightarrow 4 \times F(3)$
 - $F(3) \rightarrow 3 \times 2 \rightarrow 6$
 - $F(2) \rightarrow 2 \times 1 \rightarrow 2$
 - $F(1) \rightarrow 1 \times 1 \rightarrow 1$
 - $F(0) \rightarrow 1$

$$\begin{cases} 1 & \text{se } n = 0 \\ n \times (n - 1)! & \text{se } n > 0 \end{cases}$$

Recursão

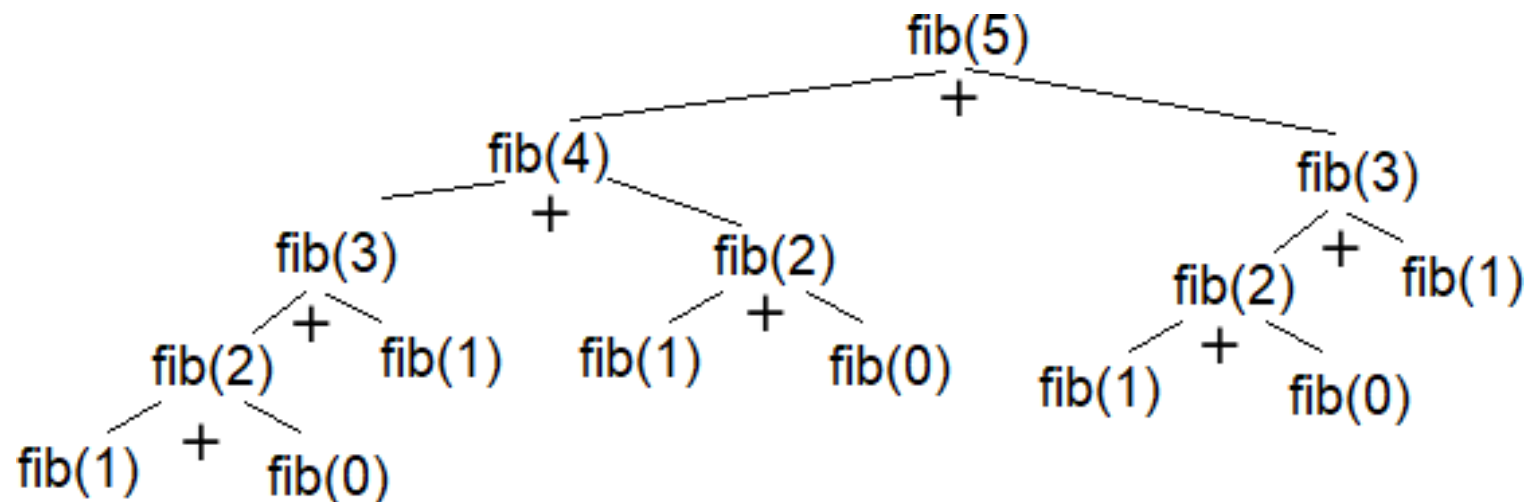
- Uma função recursiva é aquela que faz referência a si própria na sua definição
- Um exemplo clássico é o cálculo fatorial:
- $(!4) = (4 \times 3 \times 2 \times 1) \rightarrow 24$
 - $F(4) \rightarrow 4 \times 6 \rightarrow 24$
 - $F(3) \rightarrow 3 \times 2 \rightarrow 6$
 - $F(2) \rightarrow 2 \times 1 \rightarrow 2$
 - $F(1) \rightarrow 1 \times 1 \rightarrow 1$
 - $F(0) \rightarrow 1$

$$\begin{cases} 1 & \text{se } n = 0 \\ n \times (n - 1)! & \text{se } n > 0 \end{cases}$$

Recursão

- Outro exemplo é a série de Fibonacci:

$$\begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ F(n-1) + F(n-2) & \text{se } n > 1 \end{cases}$$



Recursão

- E na programação?
- $(!4) = (4 \times 3 \times 2 \times 1) \rightarrow 24$
 - $F(4) \rightarrow 4 \times F(3)$
 - $F(3) \rightarrow 3 \times F(2)$
 - $F(2) \rightarrow 2 \times 1 \rightarrow 2$
 - $F(1) \rightarrow 1 \times 1 \rightarrow 1$
 - $F(0) \rightarrow 1$

$$\begin{cases} 1 & \text{se } n = 0 \\ n \times (n - 1)! & \text{se } n > 0 \end{cases}$$

```
def fatorial(n):  
    if n == 0:  
        return 1  
    else:  
        return n * fatorial(n - 1)
```

Recursão

- E na programação?
- $(!4) = (4 \times 3 \times 2 \times 1) \rightarrow 24$
 - $F(4) \rightarrow 4 \times F(3)$
 - $F(3) \rightarrow 3 \times F(2)$
 - $F(2) \rightarrow 2 \times 1 \rightarrow 2$
 - $F(1) \rightarrow 1 \times 1 \rightarrow 1$
 - $F(0) \rightarrow 1$

$$\begin{cases} 1 & \text{se } n = 0 \\ n \times (n - 1)! & \text{se } n > 0 \end{cases}$$

```
def fatorial(n):  
    if n == 0:  
        return 1  
    else:  
        return n * fatorial(n - 1)
```

- Ao implementar recursão deve-se definir:
 - Problema Base (Critério de Parada)
 - Convergência para o problema base (sub-problema, ex: $(n-1)$)

Recursão

- Recursão vs Iteração

```
def fatorial_recursivo(n):  
    if n == 0:  
        return 1  
    return n * fatorial_recursivo(n - 1)
```

```
def fatorial_iterativo(n):  
    resultado = 1  
    for i in range(1, n + 1):  
        resultado *= i  
    return resultado
```

- Vantagens

- Os algoritmos são mais compactos e fáceis de entender
- Vasta aplicação em estruturas de dados (Ex: Árvores)

- Desvantagens da recursão

- Sobre-uso de memória (Memory Overhead)

Recursão

- Vamos codificar alguns exercícios:
 - Print N até 0 (Avaliar print antes e depois do return)
 - Soma de N até 0
 - Soma dos dígitos de um Número
 - Soma de um vetor
 - Fatorial/Fibonacci
 - Selection Sort

- Siga o roteiro
 - Encontrar o problema base
 - Definir a convergência